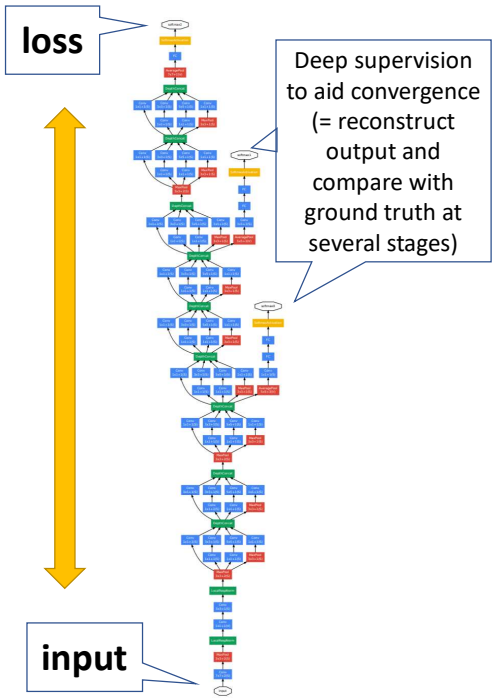
Deep Learning - BatchNorm

Bernhard Kainz

Batch Normalization

- Loss is calculated at last layer
  - Last layers learn quickly
- Data input is at first layer
  - First layers change - everything changes
  - Last layers need to relearn many times
  - Slow convergence
- This is like covariate shift...
  Can we avoid changing last layers while learning first layers?



The problem with very deep networks is that they often have a difficult time to converge.
Some tricks like deep supervision help, where you reconstruct a partial output at intermediate stages and backpropagate the loss also from there. This helps to some degree but people thought that there must be better ideas.

This is what the original idea in batch normalization was.

basically the idea is , at least that's what the intuition was when they invented it,
as I'm training my network the gradients percolate through from the top to the bottom

and so the last layers will start to adapt and so they adapt to whatever the labels are

and then the next layer down will start to adapt next layer down will start adapt and so I get this cascade of stuff that keeps on adapting.

the trouble is as I'm adapting from the top down, the features are going back up, are going to change so now that last layer that was actually fairly well
adapted to begin with, has to readapt now to the new inputs that it's getting.

so it takes a very long time to converge and adopt all layers.

...

Batch Normalization

- Can we avoid changing last layers while learning first layers?
- Fix mean and variance

$$\mu_B = \frac{1}{|B|}\sum_{i \in B} x_i \; and \; \sigma_B^2 = \frac{1}{|B|}\sum_{i \in B}(x_i - \mu_B)^2 + \varepsilon$$

and adjust it separately

$$x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$$

mean

variance

maybe we can fix things. Maybe we can fix at least something.
we can fix them by picking a given mean and a given variance and just at least correcting for that as we train these.
you don't want to completely fix those layers but at least you could say we're going to fix them up to an affine transform that we're going to learn separately.

Affine transform means multiply and add.

okay so let's just say and we pick a mean and the variance
you know mu and Sigma squared and we then renormalize the data

so we take Xi minus the mean divided by the variance or standard deviation

and then I allow for a separate coefficient gamma and a separate offset beta to take care of things

this is batch normalization.

this is the part where you should start asking questions
with the benefit of hindsight it's always really easy to ask those questions whether that

covariance shift correction was really the entire reason of what this was made for and whether that's really why it works.

Well it works, that has been shown in many experiments.

Batch Normalization

- Doesn't really reduce covariate shift (Lipton et al. 2018) https://arxiv.org/abs/1805.10694
- Regularization by noise injection

$$x_i = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Random offset

learned

learned

Random scale

  - Random shift per mini batch
  - Random scale per mini batch
- No need to add dropout (both are capacity control)
- Ideal mini batch size: 64-256

The awful truth is that the original motivation was wrong and it does not really reduce covariate shift.

there's a paper by Lipton et al. where they go and measure the contribution of BatchNorm and they find out that it actually makes covariate shift worse.

However, it works and helps convergence. WHY?

So it actually turns out that basically this is doing regularization by noise injection.

you compute a mean and the variance on a mini batch

a mini batch of maybe 64 observations and so what you're effectively doing is you're subtracting some empirical mean and that's obviously noisy

And you're dividing by some empirical standard deviation. That's obviously also noisy.

so you're basically getting a random shift and a random scale from any batch.

this is one of the reasons why if you use batch normalization you really don't need drop

out in the same network.

they kind of do similar things in terms of capacity control.

This is also the reason why batch norm is quite sensitive to mini batch size.

if you pick a mini batch that’s too large then you're not injecting enough noise and you're not regularizing enough.

if you're picking one that’s too small then basically the noise becomes too high and then you’re not converging very well.

this doesn’t matter so much for single GPU training but as soon as you go to multi-gpu it starts to matter quite immensely.

what do you do during test time?

well what you do during test time is you just fix this essentially.
so remember there are those parameters gamma and beta. They are learned so they are basically a learned scale and offset.
We fix them and for the means and variances we’re actually going to use just the running average and the large sample size mean.

Batch Normalization

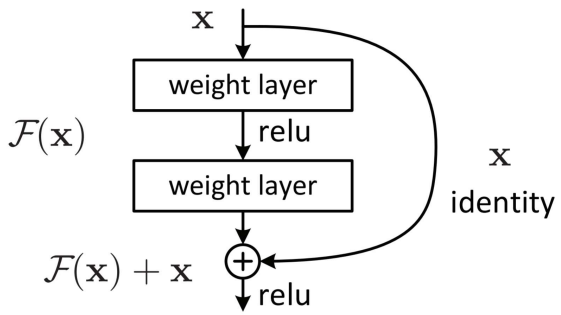
- Dense layer: One normalization for all
- Convolutional layer: One normalization per channel
- Compute **new mean and variance** for every minibatch
  - Acts as regularisation
  - Be careful when scaling up to multi-GPU training

https://xkcd.com/

if you have a dense layer then you just use one normalization for all the activations and if you have a convolution then you use one per channel.

For every mini batch a batch-norm layer computes a new mean and variance during training.

This layer requires the train flag to be set correctly in your network. Behaviour is different during testing and training.


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
x
identity
weight layer
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu